Politechnika Świętokrzyska
Wydział Elektrotechniki, Automatyki i Informatyki

Zespół:
Kot Jarosław
Prusicki Jakub

Łamacz haseł

Projekt zespołowy
na studiach stacjonarnych
o kierunku **Informatyka**
Stopień **II**

Opiekun projektu:
dr inż. Paweł Paduch

Kielce, 2023

SPIS TREŚCI

1. Haszowanie listy haseł.

Do zrealizowania projektu zdecydowano się na skorzystanie z algorytmu haszującego MD5. Pomimo mniejszego bezpieczeństwa w porównaniu do innych algorytmów, które nie jest kluczowe w tym projekcie zyskał przewagę pod względem szybkości obliczeniowej oraz prostoty - zarówno w zrozumieniu jak i samej implementacji. Ma niewielką liczbę kroków i prostą strukturę, co czyni go łatwym do użycia.

```
class MD5
{
    1 reference
    public static List<String> CreateMD5(List<String> listOfPasswords)
    {
        List<string> passwordConvertedOnMD5 = new List<string>();

        foreach (string password in listOfPasswords)
        {
            passwordConvertedOnMD5.Add(CreateMD5(password));
        }

        return passwordConvertedOnMD5;
    }

    2 references
    public static string CreateMD5(string password)
    {
        using (System.Security.Cryptography.MD5 md5 = System.Security.Cryptography.MD5.Create())
        {
            byte[] inputBytes = System.Text.Encoding.ASCII.GetBytes(password);
            byte[] hashBytes = md5.ComputeHash(inputBytes);

            //return Convert.ToHexString(hashBytes); // .NET 5 +

            // Convert the byte array to hexadecimal string
            StringBuilder sb = new System.Text.StringBuilder();
            for (int i = 0; i < hashBytes.Length; i++)
            {
                sb.Append(hashBytes[i].ToString("X2"));
            }
            return sb.ToString().ToLower();
        }
    }
}
```

Rys. 1. Klasa MD5.

Na podstawie działającego już kodu (sprawozdanie dot. tematu nr 4) został rozbudowany kod, który na tą chwilę umożliwia hashowanie już hashowanie listy haseł.

2. Generowanie permutacji na podstawie zakresu.

Kiedy algorytm do generowania wszystkich możliwości (permutacji z powtórzeniami) haseł na podstawie zadanego alfabetu był już gotowy, pojawiło się pytanie czy można to bardziej zoptymalizować aby nie wykonywać niepotrzebnych obliczeń. W związku z rozproszeniem obliczeń zdecydowano się generować tylko odpowiednią część możliwości. Aby to wykonać postanowiono wprowadzić ograniczenia poprzez podawanie zakresów w postaci początkowej i końcowej możliwości wygenerowanego hasła na podstawie alfabetu.

```
public class PermutationsGenerator
{
    1 reference
    public static List<string> GeneratePermutationsWithRepetitions(string alphabet,string startRange, string endRange)
    {
        List<string> permutations = new List<string>();

        GeneratePermutationsWithRepetitionsHelper(alphabet,startRange, endRange, "", permutations);

        return permutations;
    }

    2 references
    private static void GeneratePermutationsWithRepetitionsHelper(string alphabet, string startRange, string endRange, string current, List<string> permutations)
    {
        if (current.Length >= startRange.Length && CompareTwoString(current, startRange) >= 0 && CompareTwoString(current, endRange) <= 0)
        {
            permutations.Add(current);
        }

        if (current.Length >= endRange.Length)
        {
            return;
        }

        for (int i = 0; i < alphabet.Length; i++)
        {
            GeneratePermutationsWithRepetitionsHelper(alphabet, startRange, endRange, current + alphabet[i], permutations);
        }
    }
```

Rys. 2.1 Generowanie permutacji z zakresu cz1.

Podczas wstępnych założeń nie przewidywano większych trudności, jendka te pojawiły się dopiero w fazie testów. Okazało się że program działa dla zakresów, jednak nie pokazuje wszystkich wyników. Problemem była sama funkcja porównująca (**string.Compare**) , która zapewne była napisana z myślą o innych porównaniach. Jej działanie polega na kolejnym porównywaniu kolejnych odpowiadających liter w tych napisach. Problem jednak pojawia się kiedy podamy zakres, który łamie tę logikę choćby poprzez długość napisów np. BB - AAB. W takim przypadku napis AAB zostanie uznany za ‘mniejszy’, ponieważ już po porównaniu pierwszych liter program uznał, że skoro B > A to BB musi być większe od AAB. Poniżej przedstawiono funkcję porównującą napisy z uwzględnieniem długości ciągu znaków, wykorzystaną w tym projekcie (w funkcji generującej możliwe hasła).

```
    private static int CompareTwoString(string s1, string s2)
    {
        char[] charArray1 = s1.ToCharArray();
        char[] charArray2 = s2.ToCharArray();

        int minLength = Math.Min(charArray1.Length, charArray2.Length);

        if (charArray1.Length < charArray2.Length)
            return -1;
        else if (charArray1.Length > charArray2.Length)
            return 1;

        for (int i = 0; i < minLength; i++)
        {
            if (charArray1[i] < charArray2[i])
                return -1;
            else if (charArray1[i] > charArray2[i])
                return 1;
        }

        return 0;
    }

}
```

Rys. 2.1 Generowanie permutacji z zakresu cz.2 - funkcja porównująca.